Università degli studi di Parma
Dipartimento di Ingegneria e Architettura

Sistemi operativi e in tempo reale – a.a. 2022/23

Altri esercizi di programmazione concorrente mediante Monitor

Barbiere addormentato

- Descrizione del problema (da *Modern Operating Systems*, A. Tanenbaum):

  Un negozio di barbiere ospita, oltre al barbiere stesso, una sedia reclinabile su cui viene fatto accomodare il cliente durante il servizio ed N sedie per i clienti in attesa. In assenza di clienti, il barbiere si sistema sulla sedia reclinabile e si addormenta. Ogni cliente che arriva quando il negozio è vuoto deve quindi provvedere a risvegliare il barbiere addormentato. I clienti che arrivano con il barbiere già al lavoro si accomodano sulle sedie, se ci sono posti disponibili, oppure lasciano il negozio, se tutte le sedie sono occupate.
  Si risolva il problema mediante il costrutto monitor prestando particolare attenzione ad evitare situazioni di "corsa" nell'interazione tra barbiere e clienti.

Barbiere addormentato

- Alcune caratteristiche del problema:
    - relazione cliente-servitore
    - *rendezvous* tra servitore e generico cliente
    - sequenza di passi sincronizzati che iniziano con un rendezvous
    - rischio di situazioni di "corsa"

- Le entry dei thread:

clienti	barbiere
...	*repeat*
b_shop.get_haircut;	b_shop.get_next_customer;
...	<cut hair>
	b_shop.finish_customer;
	forever;

- Nella variante più nota del problema (v. testo) la sala d'attesa ha una capienza limitata

- Condizioni di sincronizzazione:

    - clienti in attesa del servizio da parte del barbiere
    - cliente che viene servito, in attesa che il barbiere lo faccia uscire
    - barbiere in attesa dell'arrivo di un cliente
    - barbiere in attesa dell'uscita del cliente (opzionale)

Barbiere addormentato

```
var   b_shop:     barbery;     /* il tipo e' definito successivamente */

Customer i:
repeat       /* once in a while */
       var   done          boolean;

       done := false;              /* get_haircut  */
       repeat until  done
              b_shop.enter_shop (done);
              if not  done then  sleep (30 min);
                     /* waste random amount of time and come back later */
       end
       < do useful things >      /* for about 1 month */
forever


Barber:
repeat

       b_shop.get_next_customer;
       service;                          /* fuori dal monitor */
       b_shop.finish_customer;
forever
```

Barbiere addormentato

```
#define  Nchairs          5
type  barbery = monitor;
     var      waiting_c:                              0 .. Nchairs;     /* customers */
              barber_busy, chair_occupied, finished:          boolean;
              barber_ready, customer_ready,
              end_of_service, exited_customer:                condition;



     procedure entry  enter_shop (var  served: boolean);
      begin
              if  waiting_c = Nchairs  then  served := false;
              else begin
                            waiting_c := waiting_c + 1;
                            while  barber_busy  do  barber_ready.wait;
                            waiting_c := waiting_c - 1;
                            chair_occupied := true;     // oppure l'id del cliente
                            customer_ready.signal;
                            while  (not  finished)  do  end_of_service.wait;
                            finished := false;
                            exited_customer.signal;
                            served := true;
                     end
      end
```

Barbiere addormentato

// segue
// la gestione dello stato del servitore (barber_busy) evita problemi di “corsa”

```
    procedure entry  get_next_customer;
     begin
          barber_busy := false;
          barber_ready.signal;
          while  (not  chair_occupied)  do  customer_ready.wait;
          chair_occupied := false;
          barber_busy := true;
     end


    procedure entry  finish_customer;
     begin
          <prende il denaro e rilascia ricevuta>
          finished := true;
          end_of_service.signal;
          while  finished  do  exited_customer.wait;
     end


    begin
          waiting_c := 0;  barber_busy := true;       // ← inizializz. “busy”
          finished := chair_occupied := false;
     end
end;  /* monitor */
```

- Attenzione alla semantica ....

Barbiere addormentato

- L'interazione tra thread di questo problema è paradigmatica delle interazioni cliente-servitore:
  - se il servitore non ha nulla da fare *deve essere o rimanere sospeso*
  - il cliente che quando arriva trova il servitore sospeso (perché inattivo), *deve risvegliarlo*, comunicare la propria richiesta, e *sospendersi in attesa* del completamento del servizio
  - se il *servitore è occupato*, il cliente deve lasciare traccia della propria presenza o della propria richiesta
  - il servitore occupato, una volta completato il servizio precedente, esamina la coda di richieste pendenti, e se non è vuota inizia un nuovo servizio
  - spesso è utile che il cliente selezionato venga fatto avanzare di stato (si risospende poi immediatamente) per separare i clienti in attesa da quello che viene servito
  - da notare l'inizializzazione dello stato del servitore: sempre ad occupato per prevenire problemi di "corsa" iniziale;
  - il servitore si dichiara *pronto all'inizio della entry in cui accetta una nuova richiesta* e non al termine della entry in cui completa il servizio della richiesta precedente, in modo da prevenire situazioni di "corsa" nel funzionamento a regime.

Sistemi operativi e in tempo reale

Prova scritta del ...

Nome e Cognome: …….. …… Matricola: ……
Filename dell'elaborato per l'Esercizio 1: …..…… Firma: …….
Postazione di lavoro: ……..

Note:
- Compilare, firmare e consegnare questo foglio assieme ai fogli con la risposta agli Esercizi successivi.
- Salvate frequentemente il vostro elaborato. Il nome del file deve essere del tipo "cognome.c".
- Inserite all'inizio del file i vostri dati (Nome, Cognome, matricola).

E1	TOT1				E2	TOT2			
	20					2			

Esercizio 1: (20 punti)
Questo esercizio deve essere svolto all'elaboratore tramite editor. Utilizzate una sintassi pseudo-C. La seconda parte del compito non viene corretta se questo esercizio non è svolto o è insufficiente.

Pizzeria

E' stata aperta in zona una pizzeria che vende anche per asporto e che presenta prezzi molto competitivi. La pizzeria è gestita da un unico pizzaiolo che si occupa anche di ricevere le ordinazioni dei clienti. Nel locale sono disponibili solo K posti, e pertanto chi intende consumare sul posto deve accettare di accomodarsi nel primo posto disponibile. Per lo stesso motivo, chi consuma nel locale viene servito individualmente (non sono cioè gestiti i gruppi di più persone) e si accomoda al tavolo dopo aver ricevuto la pizza.
Per l'ordinazione, i clienti si presentano in due code distinte (A=asporto e P=consumazione sul posto). Il pizzaiolo riceve una sola ordinazione per volta e procede direttamente a preparare quanto richiesto. La pizza o le pizze ordinate vengono consegnate al cliente prima di accettare la successiva ordinazione (non sono cioè mescolate le preparazioni di pizze di ordinazioni diverse). Il cliente che ha richiesto pizze per asporto, dopo avere presentato l'ordinazione, attende in un angolo del locale, obnubilato dai fumi, di ricevere le pizze. Altrettanto fa il cliente che intende consumare sul posto. Un cartello precisa che in presenza di tavoli liberi vengono serviti prima i clienti che consumano sul posto, mentre in assenza di tavoli liberi non è possibile servire chi vuole consumare sul posto. Per il bere, vale la politica BYO (bring your own).

Si risolva il problema di programmazione concorrente utilizzando il costrutto monitor, indicandone in modo esplicito la semantica. Si evitino deadlock, attese inutili e attese attive. Non è consentito l'impiego delle primitive `broadcast` e `queue` sulle variabili condizione.

Esercizio 2: (2 punti)
L'esercizio è da svolgere su questo foglio e verrà corretto solo se l'Es.1 è sufficiente.
// Domandina di teoria da svolgere sul retro del foglio. Serve per integrare il punteggio.

Inizio definizione del tipo ed entry del thread servitore

```
//Risoluzione con monitor in semantica Mesa

#define K <numero tavoli>


type pizzeria=monitor;
begin
        integer: clientiA, clientiP, posti_liberi;
        condition: codaA, codaP, attendi_clienti, attendi_ordinazione, attendi_servizio;
        boolean: tavoli[K], ok_ordina, ordinazione, servito;


        procedure entry accetta_cliente();     // processo servitore
        begin
                while (clientiA == 0 && (posti_liberi==0 || clientiP==0)) do
                begin
                        attendi_clienti.wait();
                end;
                ok_ordina=TRUE;
                if (clientiP>0 && posti_liberi>0) then codaP.signal();
                else codaA.signal();
                ordinazione=FALSE;
                servito=FALSE;
                while (!ordinazione) do
                begin
                        attendi_ordinazione.wait();
                end;
        end;


        procedure entry servi_cliente();
        begin
                servito=TRUE;
                attendi_servizio.signal();
        end;
```

Entry (unica) del cliente di tipo A

```
procedure entry ordina_pizza();    // clienti per Asporto
begin
        if (clientiA==0) then attendi_clienti.signal();
        clientiA++;
        while (!ok_ordina) do
        begin
                codaA.wait();
        end;
        clientiA--;
        ok_ordina=FALSE;
        ordinazione=TRUE;  //specifica dell'ordine
        attendi_ordinazione.signal();
        while (!servito) do
        begin
                attendi_servizio.wait();
        end;
        servito=FALSE;
end;
```

Entry del cliente di tipo P (perchè due entry?)

```
procedure entry entra(int posto_occupato);     // clienti sul Posto
begin
        if (clientiP==0) then attendi_clienti.signal();
        clientiP++;
        while (!ok_ordina || posti_liberi==0) do  // in realtà la verifica se ci sono
        codaP.wait();                             // posti è superflua

        clientiP--;
        posti_liberi--;  //prenota già un posto
        ok_ordina=FALSE;
        ordinazione=TRUE;
        attendi_ordinazione.signal();
        while (!servito) do  attendi_servizio.wait();
        servito=FALSE;
        for (int i=0;i<K;i++)   // per questo esercizio il vettore tavoli non serve
        begin                   // basta contatore intero Nposti_occupati
                                // in altri esercizi occorre tenere traccia occupazione
                if(tavoli[i]==0) then      // esempio passaggio parametro in uscita
                begin
                        posto_occupato=i;
                        tavolo[i]=1;
                        break;
                end;
        end;
end;
// in alternativa al passaggio per indirizzo:    return (posto_occupato);


procedure entry esci(int posto_occupato);
begin
        tavoli[posto_occupato]=0;       // deve liberare il posto, avendo previsto
        posti_liberi++;                 // vettore il "tavolo" da gestire
        attendi_clienti.signal();       // evita attesa inutile eventuale cliente P
end;                            // -> avvisa il pizzaiolo, che verificherà le condizioni
```

Inizializzazione (inclusa entro definizione del tipo)

```
begin    //inizializzazione

        clientiA=0;
        clientiP=0;
        posti_liberi=K;
        ok_ordina=FALSE;
        ordinazione=FALSE;
        servito=FALSE;
        for (int l=0;l<K;l++)
        begin
                tavoli[l]=0;
        end;
end;
```

end. //fine della dichiarazione del monitor, ovvero della definizione dell'ADT

Creazione istanza e tracce thread con uso delle entry

```
//creazione istanza del monitor
pizzeria : Langhirano;


//traccia processo A (asporto)
...
Langhirano.ordina_pizza();          // chiede servizio, può attendere in vari stati
...


//traccia processo P (consumazione sul posto)
int tavolino;
...
Langhirano.entra(tavolino);     //passaggio parametri di uscita per indirizzo
<mangia>                        // non può mangiare nella entry!
Langhirano.esci(tavolino);
...


//traccia processo pizzaiolo
while (1) do                        // server, fa sempre questo
begin
      Langhirano.accetta_cliente();
      <prepara pizza>               // attività fisica, durata ≠ 0  -> non nella entry!
      Langhirano.servi_cliente();
end;
```

Nota: Assieme alla definizione del tipo di dato astratto (il "monitor") consegnate sempre anche la traccia di esecuzione di tutti i tipi di processi / thread e la creazione dell'istanza. E' il punto di partenza della correzione…

Altre osservazioni generali

- Le attività “fisiche” che richiedono tempo devono *sempre essere esterne alle procedure entry*
  - Esempi di attività fisiche: mangiare, guidare un veicolo, camminare, visitare, preparare la pizza, tagliare i capelli, etc.
  - Le procedure entry servono solo per le sincronizzazioni e le attese
- I clienti che *ricevono un servizio sono sempre sospesi su una condizione*
  - Esempi di servizi ricevuti: taglio di capelli, attendere la pizza, essere trasportati da un guidatore
  - Diversamente da quanto accade nel mondo reale, i thread che ricevono un servizio devono essere sospesi su una condition
  - Utilizzate una condition specifica per i clienti che *stanno ricevendo un servizio*, distinta dalla condition su cui attendono i clienti quando sono *in attesa che il loro servizio inizi*: la condizione di attesa taxi è diversa da quella su cui attende il cliente durante il servizio di trasporto
- Non prevedere mai, cioè mai, un’attesa attiva, né interna né esterna alla procedura entry

Boat pooling

- Il problema:

Argomento della prova è il *boat pooling,* una tecnica per ridurre l'inquinamento da traffico mediante condivisione dei mezzi di trasporto utilizzata nell'area di Redmond, WA.

Per l'attraversamento di un fiume è disponibile una barca a 4 posti. La barca è utilizzata da dipendenti della Microsoft e da hacker di Linux. In ogni attraversamento la barca può partire solo se al completo, ma le combinazioni dell'equipaggio con tre passeggeri di un tipo ed uno solo dell'altro sono proibite per motivi di ordine pubblico.

I 4 passeggeri salgono sulla barca quando essa arriva o immediatamente se essa è già disponibile. Quando anche l'ultimo passeggero è salito sulla barca ha inizio il viaggio, che termina con il ritorno della barca. I passeggeri devono intendersi scaricati solo al ritorno della barca.

Si risolva il problema di programmazione concorrente utilizzando il costrutto monitor, indicandone la semantica. Si provveda sia alla definizione del tipo, sia alla creazione ed utilizzo dell'istanza da parte dei processi. Si prevedano distinte procedure entry per la gestione dei due tipi di passeggeri, modellando esplicitamente lo stato dei processi `LinuxHacker` e `MicrosoftEmployee` all'interno del monitor. Si osservi che i due tipi di processi "cliente" possono essere bloccati in attesa dell'inserimento in un equipaggio, dell'imbarco ed inizio del viaggio e durante il viaggio stesso. Una possibile traccia di riferimento è la seguente:

```
        LH:                                     ME:
< hack(); >                             < fix_bugs(); >
river_monitor.LH_want_to_board();       river_monitor.ME_want_to_board();
< more_hack(); >                        < more_fixes(); >

        boat:
repeat
        river_monitor.board_passengers();
        <cross river>;
        river_monitor.return_boat();
forever
```

Si evitino situazioni di attesa attiva e di attesa inutile. Non è consentito l'impiego delle primitive `broadcast` e `queue` sulle variabili condizione. Rispettando le specifiche del problema, in quale modo le azioni entro il monitor degli hacker Linux e dei dipendenti Microsoft possono differire?